Iterators and Generators

New ways of looping in JavaScript

Iterables

Built-in Objects that are already Iterable.

- Array
- Map
- Set
- NodeList
- HTMLCollection
- And many others

Plain Objects are NOT iterable.

Iteration Protocols

Iterator Protocol - Object provides a next() function that returns an object with two keys, value and done.

Iterable Protocol - Has a property with the key of "`@@iterator`" (available via the global `Symbol.iterator`) that is a function which returns an object conforming to the Iterator protocol.

Why do we care?

These protocols make possible two pieces of JavaScript functionality.

`for...of` loops

and

`Array` spread and destructuring

Let’s try it out on the built-ins

A note on `for...in`

The `for...in` loop ***does not*** use the Iteration protocols. It does work on regular JavaScript Objects, to loop through the keys of an Object.

When using Arrays, Maps, Sets, and NodeLists, you want to stick to `for...of`.

However `for...of` does not work with plain Objects.

Let’s make our own Iterable objects

Generators

An easier way to build iterable objects.

```
function* generator() {
  yield 1;
  yield 2;
  yield 3;
}
```

Properties of Generators

Generator Objects are created by a generator function by using the special `function*` syntax.

They conform to both the iterable and iterator protocols, which means we can use them with `for...of` and spread/destructuring

Calling the generator function returns you a generator object, but the function only runs until it reaches the first `yield` keyword

The `yield` keyword causes the generator object to stop running the function (much like await causes an async function to pause).

Calling `.next()` on the generator causes the function to continue past the yield, until it reaches a new yield or the function finishes.

Let’s build a generator

Async Iterables

No built in object implement async iterables.

Allows you to use the special loop syntax `for await...of`

Uses promises. Can also just use async/await.

You can build them manually using the `Symbol.asyncIterator` key, but it's much easier to take advantage of async generators.

Async Generators

Just a generator function that is also an async function.

```
async function* generator() {
    yield 1;
    yield 2;
    yield 3;
}
```

Let’s code an async generator